```
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Module: Resident
;>
;>      This module contains all the routines ( besides the Z8 initialization
;>      procedures ) that must be resident within the Z8.
;>
;>      RESIDENT PROCEDURE Bsy_Clr( MemAdr : PTR { !!Wrk_lo+C } )
;>                                  Response     : BYTE { Wrk_lo+$A }
;>                                  Cmnd_Pending : BIT { Wrk_lo+$B/Bit 7 }
;>                                  !Bsy         : BIT { Wrk_lo+$B/Bit 6 }
;>                                )
;>      RESIDENT PROCEDURE Wait_Cmd( Cmnd_Pending : BOOLEAN { !rB/bit 7 }
;>                                   Response : BYTE { !rA }
;>                                   !Bsy     : BOOLEAN { !rB/bit 6 }
;>                                   MemAdr   : PTR { !!rC }
;>                                 )
;>      RESIDENT PROCEDURE Get_Wr_Data( Response : BYTE { !r9 } )
;>      RESIDENT PROCEDURE Ack_Read( Response )
;>      RESIDENT FUNCTION Wr_Resident : Status : BYTE { !r0 }
;>      RESIDENT FUNCTION Fmt_Resident : Status : BYTE { !r0 }
;>      RESIDENT FUNCTION RdHdr_Resident : Status : BYTE { !r0 }
;>      RESIDENT FUNCTION Rd_Resident : Status : BYTE { !r0 }
;>      RESIDENT SubFUNCTION Start_StMach
;>      FUNCTION Sub3( A, B : 3 BYTES { !r0:2, !rC:E } ) : 3 BYTES { !r0:2 }
;>      FUNCTION Add3( A, B : 3 BYTES { !r0:2, !rC:E } ) : 3 BYTES { !r0:2 }
;>      PROCEDURE Set_RamBank( Ram_Bank : BYTE { !r0 } )
;>      PROCEDURE Set_Led( State : BIT { !r0/bit 0 } )
;>      FUNCTION LoadStatus : BYTE { !r0 }
;>      PROCEDURE SetStatus( StatusByte : BYTE { !r0 }
;>                           Value : BYTE { !r1 }
;>                         )
;>      PROCEDURE Set_Dmt( Parent1, Parent2 : BYTE { !rB, !r0 } )
;>      PROCEDURE Clr_Dmt
;>      FUNCTION FormatBlock( Paraent : BYTE { !r8 } ) :
;>                            BOOLEAN
;>                            Status : BYTE { !r0 }
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Bsy_Clr
;>
;>      Inputs:
;>              Response     : BYTE { Wrk_lo+$A }
;>              Cmnd_Pending : BIT { Wrk_lo+$B/Bit 7 }
;>              !Bsy         : BIT { Wrk_lo+$B/Bit 6 }
;>              MemAdr       : PTR, { !!rC }
;>
;>      Outputs: { none }
;>
;>      BEGIN
;>       WHILE CMD DO BEGIN END { busy wait on CMD }
;>       Set up memory select: Apple <--> Mem
;>       Memory_Address_Register := MemAdr
```

```
;>       Reset BSY
;>       WHILE NOT( CMD ) DO BEGIN END { busy wait on CMD- }
;>       JUMP( Start_Command ) { begin a new command }
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Clr_Bsy:
                Srp     #Wrk_lo

                Ld      !Port2,#Not_StartL+Disk_Mem+DrwL_Read+Bsy
                Lde     @!!rC,!rC

                Ld      P3m,!P3m_StMach
                And     !Port3,!Dm_Mask ;set DM/loPort to Low
                Ld      P01m,!P01m_StMach
                Ld      !Port2,#Not_StartL+Disk_Mem+DrwL_Read+Bsy ;toggle AOE
                Tm      !rB,#MultiWr ;check for mutli-write command
                Jr      Z,Clr_Bsy_Rd

                Ld      !Port2,#Not_StartL+Apple_Mem
                Jr      Clr_Bsy1

Clr_Bsy_Rd:     Ld      !Port2,#Not_StartL+Apple_Mem+DrwL_Read

Clr_Bsy1:       Ld     !rF,#Cmd                ;load mask
                Ld     !rC,#.HIBYTE. Cmnd_Ptr ;MemAdr := Command Buffer
                Ld     !rD,#.LOWBYTE. Cmnd_Ptr


Bsy_Lp2:        Tm      !Port2,!rF              ;test for CMD active hi
                Jr      Z,Bsy_Lp2

;               \ /   In_Line code, done for SPEED!
                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Wait_Cmd
;>
;>      Inputs:
;>            Cmnd_Pending : BOOLEAN, { !rB/bit 7 }
;>            Response     : BYTE, { !rA }
;>            !Bsy         : BOOLEAN, { !rB/bit 6 }
;>            MemAdr       : PTR, { !!rC }
;>
;>      Outputs: { none }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Set up memory select: Apple <--> Z8
;>       WHILE NOT( CMD ) DO BEGIN END { busy wait on CMD- }
;>       Write( Response ) to Apple
;>       Set BSY
```

```
;>       WHILE CMD DO BEGIN END { busy wait on CMD }
;>       Read( Apple_Resonse ) from Apple
;>       IF ( Apple_Response = $55 )
;>        THEN
;>         IF NOT( IBsy )
;>          THEN
;>            Memory_Address_Register := MemAdr
;>            Reset BSY
;>            WHILE NOT( CMD ) DO BEGIN END { busy wait on CMD- }
;>         Set up memory select: Z8 <--> Mem
;>         IF Cmnd_Pending
;>          THEN RETURN
;>          ELSE JUMP( Start_Command ) { begin a new command }
;>        ELSE ABORT( Apple_Interface_Exception, Cmnd_Pending,
;>                    Response, Apple_Response )
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Wait_Cmd:
                Ld      P01m,!P01m_Image ;restore port to load adr
                Ld      !Port2,#Not_StartL+Disk_Mem+DrwL_Read
                Lde     @!!rC,!rC ;set external memory address

Wait_Cmd1:      Ld      !Port2,#Not_StartL+Z8_Apple+DrwL_Read

                Ld      P01m,#P0_03_Adr+P0_47_Out+Stack_In+P1_Out

                Ld      !Port1,!rA              ;Apple gets Response

                Ld      !Port2,#Not_StartL+Z8_Apple+DrwL_Read+Bsy

                Ld      !rF,#Cmd
                Ld      !rE,#Apl_Ack    ;get ready for response
                Ld      !rD,#!Bsy

Cmd_Lp2:        Tm      !Port2,!rF              ;test for CMD inactive low
                Jr      Nz,Cmd_Lp2

;Make Port1 an input
                Ld      P01m,#P0_03_Adr+P0_47_Out+Stack_In+P1_In
                Ld      !Port2,#Not_StartL+Disk_Mem+DrwL_Read+Bsy ;strobe AOE
                Ld      !Port2,#Not_StartL+Z8_Apple+DrwL_Read+Bsy ;strobe AOE

                Cp      !Port1,!rE
                Jr      Nz,Cmd_NotAck

Cmd_TstBsy:     Tm      !rB,!rD                 ;test !Bsy
                Jr      Z,No_!Bsy

;get back to System access
Cmd_Leave:      Ld      P01m,!P01m_Image
                Ld      !Port2,#Not_StartL+Bsy+Z8_Mem+DrwL_Read
                Ld      P3m,!P3m_Image
                Srp     #Wrk_Sys
                Ret                             ;resume command
```

```
;MEM <--> Host(wr)

;change DM to output
No_!Bsy:        Ld      P3m,!P3m_StMach
                And     !Port3,!Dm_Mask ;set DM/ioPort low

                Ld      !Port2,#Not_StartL+Apple_Mem

                Ld      !rF,#Cmd
Cmd_Lp3:        Tm      !Port2,!rF              ;test for CMD active hi
                Jr      Z,Cmd_Lp3

                Ld      P01m,!P01m_Image
                Ld      !Port2,#Not_StartL+Z8_Mem+DrwL_Read
                Ld      P3m,!P3m_Image
                Tm      !rB,#Cmnd_Pending
                Jp      Z,Start_Vector
                Ret


Cmd_NotAck:     Ld      !rF,!Port1
                Ld      P01m,!P01m_Image
                Ld      P3m,!P3m_Image
                Ld      !Port2,#Not_StartL+Bsy+Z8_Mem+DrwL_Read
                Srp     #Wrk_Sys

                Cp      Wrk_Io+$F,#Free_Proc ;check for Freeing up the bus
                Jp      Z,Free_Vector

Wt_Cmd_Nak:     Push    Wrk_Io+$F
                Call    ClrNormStat
                 Clr     !r0 ;byte 0
                 Ld      !r1,#Bad_55
                Call    SetStatus
                 Pop     !r9
                Call    Abort

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>              RESIDENT ROUTINE  { source code is in Resident.Assem }
;>
;>      Procedure: Get_Wr_Data  { Get Write Data from Host }
;>
;>      Inputs:
;>            Response      : BYTE, { Wrk_Io+$A }
;>
;>
;>      BEGIN
;>       Wait_Cmd( Command_Pending, Response, NotIBsy, WBuffer1 )
;>       Wait_Cmd( Command_Pending, EndWriteResponse, IBsy, WBuffer1 )
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
```

```
                .FIN

Get_Wr_Data:
                Srp     #Wrk_Io

                 Ld     !rB,#Cmnd_Pending ;Cmnd_Pending := True, !Bsy := False
                 Ld     !rC,#.HIBYTE. WBuffer1
                 Ld     !rD,#.LOWBYTE. WBuffer1
                Call    Wait_Cmd

                 Ld     !rA,#End_Wr_Response
                 Ld     !rB,#Cmnd_Pending+!Bsy ;Cmnd_Pending := True
                Call    Wait_Cmd

                Srp     #Wrk_Sys
                Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>              RESIDENT ROUTINE  { source code is in Resident.Assem }
;>
;>      Procedure: Ack_Read     { Acknowledge read command, set BSY }
;>
;>      Inputs:
;>              Response : BYTE { Wrk_Io+$A }
;>
;>      Outputs: { none }
;>
;>      BEGIN
;>       Wait_Cmd( Command_Pending, Response, Bsy, x )
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Ack_Read:
                Srp     #Wrk_Io

                 Ld     !rB,#Cmnd_Pending+!Bsy
                Call    Wait_Cmd1

                Srp     #Wrk_Sys
                Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: Wr_Resident  { Write Resident }
;>
;>      This function exists primarily because the architecture of the
;>      Widget Controller prevents the Z8 from executing instructions
```

```
;>      from external program memory while thr state machine is running.
;>
;>      Inputs: { none }
;>
;>      Outputs:
;>              Status: BYTE { !r0 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Msel0:1 := Disk <--> Mem
;>       Set-up external ram address counter for WRITE
;>       DiskRW := Write
;>       RdHdrH := False
;>       IF ( Cylinder > RWI_Cylinder )
;>        THEN
;>              PC := True
;>              RWI := True
;>        ELSE
;>              PC := False
;>              RWI := False
;>       StartStateMachine
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN


Wr_Resident:    Srp     #Wrk_lo

                Ld      !Port2,#Not_StartL+Bsy+Disk_Mem

                 Ld     !rE,Hi_Rwi_Reg
                 Ld     !rF,Lo_Rwi_Reg
                 Sub    !rF,Cylinder+1 ;check for  ( > RWI_Cylinder )
                 Sbc    !rE,Cylinder
                Jr      Lt,WPC_Else

                Ld      !rA,#$7F ;bits inactive
                Jr      WPC_End

WPC_Else:       Ld      !rA,#$7F-RWI-PC

WPC_End:        Jr      Rd_Res2

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: Fmt_Resident  { Format Resident }
;>
;>      This function exists primarily because the architecture of the
;>      Widget Controller prevents the Z8 from executing instructions
;>      from external program memory while thr state machine is running.
;>
;>      Inputs: { none }
```

```
;>
;>      Outputs:
;>              Status: BYTE { !r0 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Msel0:1 := Disk <--> Mem
;>       Set-up external ram address counter for FORMAT
;>       DiskRW := Write
;>       RdHdrH := True
;>       IF ( Cylinder > RWI_Cylinder )
;>        THEN
;>              PC := True
;>              RWI := True
;>        ELSE
;>              PC := False
;>              RWI := False
;>       StartStateMachine
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN


Fmt_Resident:   Srp     #Wrk_lo

                Ld      !Port2,#Not_StartL+Bsy+Disk_Mem

                 Ld     !rE,Hi_Rwi_Reg
                 Ld     !rF,Lo_Rwi_Reg
                 Sub    !rF,Cylinder+1 ;check for  ( > RWI_Cylinder )
                 Sbc    !rE,Cylinder
                Jr      Lt,FPC_Else

                Ld      !rA,#$FF ;bits inactive
                Jr      FPC_End

FPC_Else:       Ld      !rA,#$FF-RWI-PC

FPC_End:        Ld      !rE,#.HIBYTE. FormatArray
                Ld      !rF,#.LOWBYTE. FormatArray

                Jr      Start_StMach

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: RdHdr_Resident  { Read Header Resident }
;>
;>      This function exists primarily because the architecture of the
;>      Widget Controller prevents the Z8 from executing instructions
;>      from external program memory while thr state machine is running.
;>
;>      Inputs: { none }
```

```
;>
;>      Outputs:
;>              Status: BYTE { !r0 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Msel0:1 := Disk <--> Mem
;>       Set-up external ram address counter for READ
;>       DiskRW := Read
;>       RdHdrH := True
;>       RWI := False
;>       PC := False
;>       StartStateMachine
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

RdHdr_Resident:
                Srp     #Wrk_lo

                Ld      !rA,#$FF ;bits inactive

                Jr      Rd_Res1

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: Rd_Resident  { Read Resident }
;>
;>      This function exists primarily because the architecture of the
;>      Widget Controller prevents the Z8 from executing instructions
;>      from external program memory while thr state machine is running.
;>
;>      Inputs: { none }
;>
;>      Outputs:
;>              Status: BYTE { !r0 }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Msel0:1 := Disk <--> Mem
;>       Set-up external ram address counter for READ
;>       DiskRW := Read
;>       RdHdrH := False
;>       PC := False
;>       RWI := False
;>       StartStateMachine
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
```

```
                .DO     Internal
                .LSTON
                .Page
                .FIN


Rd_Resident:
                Srp     #Wrk_lo
                Ld      !rA,#$7F ;bits inactive

Rd_Res1:        Ld      !Port2,#Not_StartL+Bsy+Disk_Mem+DrwL_Read

Rd_Res2:        Ld      !rE,#.HIBYTE. ReadArray
                Ld      !rF,#.LOWBYTE. ReadArray

;               \ /  ;In-Line code for Speed!!
                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      SubFunction: Start_StMach  { StartStateMachine }
;>
;>      This is a routine that is shared by all the resident state machine
;>      routines and exists primarily for the purpose of saving space. It's
;>      function is to complete the set-up for the state machine and then
;>      start it up and wait for it to finish.
;>
;>      BEGIN
;>       Z8 Port 3, Bit 4 := 0  { instead of DM it is an I/O port }
;>       Z8 Port 1 := Input  { keep Z8 from conflicting with Disk data }
;>       SectorsRead := 2 * NumberOfSectors
;>       WHILE SectorMark DO BEGIN END
;>       StartL := True
;>       WHILE NOT( SectorDnL ) AND ( SectorsRead <> 0 ) DO
;>             BEGIN
;>                     IF SectorMark
;>                       THEN
;>                             SectorsRead := SectorsRead - 1
;>                             WHILE SectorMark DO BEGIN END
;>             END
;>       Z8 Port 3, Bit 4 := DM
;>       Z8 Port 1 := Address/Data
;>       MselO:1 := Z8 <--> Mem
;>       Status := Status_Port
;>       IF EccErr
;>        THEN Status.CrcError := True
;>       StartL := False
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Start_StMach:
                Lde     @!!rE,!rE       ;set buffer address

                Ld      P3m,!P3m_StMach
```

```
                And     !Port3,!Dm_Mask  ;set DM/IoPort Low

                Ld      P01m,!P01m_StMach
                Ld      !Port0,!rA

St_Res_1:       Tm      !Port3,#SectorMark  ;test for sector mark
                Jr      Nz,St_Res_1

                And     !Port2,!Start_Mask        ;start state machine

                Ld      !rA,#Not_StartL
                Ld      !rB,#Not_EccErr
                Ld      !rE,#SectDnL    ;load mask
                Ld      !rF,#SectorMark
                Ld      Wrk_Sys+$A,#( NbrSctrs+2 ) ;timeout after 21 sectors
                Call    Set_Dmt

St_Res_2:       Tm      !Port3,!rF              ;count sector marks
                Jr      Z,St_Res_3
                Dec     Wrk_Sys+$A
                Jr      Z,St_Res_4

St_Res_25:      Tm      !Port3,!rF
                Jr      Nz,St_Res_2_5   ;wait for mark to go away

St_Res_3:       Tcm     !Port3,!rE  ;wait for state machine to finish
                Jr      Z,St_Res_2
                Tcm     !Port3,!rE  ;sample it twice
                Jr      Z,St_Res_2

St_Res_4:       Ld      P01m,!P01m_Image
                Ld      !Port2,#Bsy+Z8_Mem+DrwL_Read
                Ld      P3m,!P3m_Image

                Ld      !rC,#.HIBYTE. StatusPort
                Ld      !rD,#.LOWBYTE. StatusPort
                Lde     !rF,@!!rC

                Tm      !Port2,!rB
                Jr      Z,Res_EccErr

Res_StMach:     Or      !Port2,!rA       ;reset state machine
                Di                       ;clear the dead man timer

                Srp     #Wrk_Sys
                Ld      !r0,Wrk_Io+$F ;return StMach status
                Ret


Res_EccErr:     Tm      DiskStat,#Wr_Op
                Jr      Nz,Res_StMach
                And     !rF,#$FF-WrtNvIdL ;if ECC error
                Jr      Res_StMach

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Function: Sub3, Add3
;>
;>              { RESIDENT ROUTINES!! }
```

```
;>
;>      These functions perform the following 24 bit arithmetic operations:
;>
;>        Sub3: A <-- A - B
;>        Add3: A <-- A + B
;>
;>      Inputs:
;>              A: 3 BYTES { !r0, !r1, !r2 }
;>              B: 3 BYTES { !rC, !rD, !rE }
;>
;>      Outputs:
;>              A: 3 BYTES { !r0, !r1, !r2 }
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Sub3:
                Sub     !r2,!rE
                Sbc     !r1,!rD
                Sbc     !r0,!rC
                Ret

Add3:
                Add     !r2,!rE
                Adc     !r1,!rD
                Adc     !r0,!rC

                Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Set_RamBank
;>
;>      This procedure activates the 2k bank of ram that is indicated by
;>      the input parameter.
;>
;>      Inputs:
;>              Ram_Bank : BYTE { !r0 }
;>
;>      Outputs: { none }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       IF ( Ram_Bank > 3 ) THEN Abort
;>       deselect all ram bits
;>       select the ram bit pointed to by Ram_Bank
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
```

```
                .Page
                .FIN

Set_RamBank:    Cp      !r0,#4
                Jr      Lt,Set_RB_Start

                Call    Abort

Set_RB_Start:   Ld      !r2,#.HIBYTE. Ram_Table
                Ld      !r3,#.LOWBYTE. Ram_Table
                Rl      !r0 ;multiply index by 2
                Add     !r3,!r0 ;index into table
                Adc     !r2,#0
                Ldc     !r1,@!!r2
                Incw    !!r2
                Ldc     !r0,@!!r2

                Ld      !r2,#.HIBYTE. RamBank0 ;set the adr bits
                Ld      !r3,!r0
                Lde     @!!r2,!r3
                Inc     !r2
                Ld      !r3,!r1
                Lde     @!!r2,!r3
                Ret

Ram_Table:      .DB     0,0 ;adr 13 := 0, adr 12 := 0
                .DB     0,1 ;adr 13 := 0, adr 12 := 1
                .DB     1,0 ;adr 13 := 1, adr 12 := 0
                .DB     1,1 ;adr 13 := 1, adr 12 := 1

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Set_Led
;>
;>      This procedure changes the state of the controller LED. The
;>      state is determined by the value of State ( if State is Odd
;>      then the Led is lit, otherwise it is turned off ).
;>
;>      Inputs:
;>              State : BIT { !r0/bit 0 }
;>
;>      Outputs: { none }
;>
;>      Algorithm:
;>
;>      BEGIN
;>       Led[ State ] := State
;>       Bank_Image.Led := State
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Set_Led:
```

```
                And     Excpt_Stat,#$FF-Led_Stat
                Or      Excpt_Stat,!r0
                Ld     !r2,#.HIBYTE. Led
                Ld     !r3,!r0
               tat
                 Xor    !r0,#$FF-Led_Mask ;invert only the Led bit
                Jr     Set_Led

Led_Wait:        Ld     !r2,#.HIBYTE           Ret

                .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>      Inputs: { none }
;>
;>      Outputs:
;>              LoadStatus : BYTE { !r0 }
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                    .LSTON
                .Page
                .FIN

LoadStatus:
                Ld      !r2,#.HIBYTE. Status_Port
                Ld      !r3         .LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Se is used to set a particular bit or bits
;>      within a specific byte of standard status.
;>
;>      Inputs:
;>              StatusByte : BYTE { !r0 }
;
;>      END
;>
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTONPage
                .FIN

SetStatus:
                Ld      !r2,#.HIBYTE. CStatus0
                Ld      !r3,#.LOWBYTE. CStatus0
                Add  ,!r1
                Lde     @!!r2,!r0
                Ret

                .LSTOFF
                .DO     External
                .LST0
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Set_Dmt  { SetDeadManTimer }
;>
```

```
;>      This procedure sets the DeadManTimer
;>      occurs. The Set_Dmt routine also initializes the DeadManCounter
;>      as well as storing away information concerning the process(s)
;>      that started the Dmt.
;>
;>      Inputs: { none }
;>
;>      Outputs: { none }
;>
;>      Algorithm :
;>
;>       BEGIN
;>        Disable Interrupts { enteriParent2
;>        DeadManCounter := 0 { assume timer interrupts every 10 msecs }
;>        Enable Interrupts { leaving critical section }
;>       END
;>
;ernal
                .LSTON
                .Page
                .FIN

Set_Dmt:
                Di                                  ;enter c
                And     Irq,#$FF-Timer1 ;clear old events
                Ei
                Ret

                .LSTOFF
               .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure: Clr_Dmt  { ClearDeadManTimer }
;>
;>      This procedure clears the DeadManTimer bthe DeadManTimerCounter. This routine also
;>      enables the NormalTimer.
;>
;>      Inputs : none
;>
;>      Outputs : none
;>
;>      Algorithm :
;>;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
                .LSTOFF
                .FIN
                .DO     Internal
                .LSTON
                .Page
                .FIN

Clr_Dmt:
                Di                                  ;enter critical section
                Ret
              LSTOFF
                .DO     External
                .LSTON
                .Page
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;>
;>      Procedure Fragment: Chky for debugging purposes. I wanted a place in the
;>      code where the DM could be halted AFTER the host had released
;>      the controller as a process.
;>
```